Thanks Ibrahim,

Alright, so we’re here at the simulation part of modelsim, we can add these signals to the waveform to see how they all look like, but first let’s talk about the code that makes it run.

Okay so we start off just like we did in the TestBench part with the library and entity but the difference here is we’re defining a port and a generic INSIDE the entity, these variables are the outputs and inputs of the module to and from the testbench.

Next up, we have our architecture, where we define our own data type which can have either one of these 7 values, and a couple of signals.

Now, this is the code that does all the heavy lifting, inside a clocked process which executes every time the clock changes, we have a procedure named update traffic lights, which changes the states of the traffic light after a given amount of time seen here in the parameters.

Here once the time is up, we assign 0 to the wait time and update the state with the new state.

And now we have some if statements, first one determines if we are at a rising edge, meaning if the value of clock is changing from 0 to 1. If it is, we check the reset\_flag, if it is 0, we reinitialize all the traffic lights to Red, this is the case when we have completed a cycle of the traffic lights.

otherwise, we go into a case-when statement, similar to a switch-case statement in the C programming language.

So we check the state signal, if it is any value from the values we defined in our data type, we assign the next value to it, so for example, we have here the NorthNext value, first we update the traffic lights, then we assign the StartNorth value for 5 seconds using the update\_trafficlights procedure

\*scroll to bottom\*

And when we reach the last ‘when’ keyword here we update the value to become the first value, NorthNext, thus creating an infinite loop in a way, this is known as a finite state machine, A type of Automata